ISSN 1850-2954

IUNA

Instituto Universitario
Nacional del Arte

RIM

revista de investigación multimedia
año 3. número 3. marzo de 2011

3

IUNA

Instituto Universitario Nacional del Arte

editorial

... cabe tener en cuenta las posibilidades que pueden llegar a seguirse de los encuentros entre ciencia y las prácticas artísticas. En muchos casos, los artistas toman inspiración en los hallazgos de la ciencia, o investigan de manera creativa en algunos campos tecno-científicos (por ejemplo, y en estos momentos, la ingeniería genética, la nanotecnología, la matemática del límite, el software avanzado, la astronomía, la robótica, la inteligencia artificial, son campos en los que unos u otros artistas han encontrado inspiración directa y a partir de los que de hecho realizan alguna investigación creativa suficientemente relevante, cuando menos desde el punto de vista artístico).

(José Luis Brea, cultura _RAM, 2007)

Edición especial dedicada al proyecto PICTO "Diseño y desarrollo de aplicaciones e interfases de Realidad Aumentada destinadas a síntesis y procesamiento de audio digital", en el año del Bicentenario (1810-2010).

En el año del Bicentenario (1810-2010) hacemos entrega del tercer número de la Revista de Investigación Multimedia del Área Transdepartamental de Artes Multimediales del Instituto Universitario Nacional del Arte.

La misma se presenta como una edición especial de los documentos de avance de la primera etapa del proyecto PICTO "Diseño y desarrollo de aplicaciones e interfases de Realidad Aumentada destinadas a síntesis y procesamiento de audio digital", aprobado por la Agencia Nacional de Promoción Científica y Tecnológica - Fondo para la Investigación Científica y Tecnológica (Ministerio de Ciencia, Tecnología e Innovación Tecnológica - Presidencia de la Nación).

El proyecto parte del paradigma "continuo realidad-virtualidad" (Milgram, Kishino y Takemura, 1994) el cual ubica a los entornos virtuales en diversas categorías, cuyo rango se extiende desde la realidad física hasta la realidad virtual. Situados en este paradigma, entendemos por entornos de realidad aumentada a aquellos que logran conjugar elementos virtuales con la realidad física que nos rodea. Dentro de esta clasificación se encuadra la mayor parte de las experiencias de multimedia escénica, como las instalaciones, las *performances* y las intervenciones interactivas en las cuales realidad y virtualidad se funden.

En esta entrega volcamos los resultados de siete artículos que avanzan sobre los tópicos planteados. "Nuevas caracterizaciones de la actividad musical en el aula" de Prof. Carmelo Saitta abre un espacio de análisis sobre la vinculación música - espacio áulico. "Composición asistida en entorno PD" de los Dres. Pablo Cetta y Oscar Pablo Di Liscia, y "Medidas de similitud entre sucesiones ordenadas de grados cromáticos" del Dr. Oscar Pablo Di Liscia presentan técnicas de composición y organización del material musical en el entorno de composición en tiempo real *Pure data*. "Desarrollo de un sistema óptico para interfaces tangibles (mesa con pantalla reactiva)" y "Diseño de interface para el desarrollo de una pantalla sensible al tacto con aplicación musical" del Ing. Emiliano Causa exponen, respectivamente, el desarrollo de una mesa con pantalla sensible al tacto y el diseño de una interface de pantalla sensible al tacto (del tipo multitacto) aplicada a un editor gestual de música. "Técnicas de síntesis y procesamiento de sonido y su aplicación en tiempo real" del Lic. Matías Romero Costas repasa algunas de las técnicas de síntesis y procesamiento de sonido más utilizadas y difundidas, desde un punto de vista teórico, a través de ejemplos de aplicación, implementados en el entorno de programación Max-MSP. Por último, el artículo "Técnicas de programación vinculadas a la realidad aumentada y a las interfaces tangibles" del DCV Tarcisio Lucas Pirotta, presenta las principales funciones y bloques de programación disponibles en los paquetes de librería ARToolKit y reacTIVision, para su aplicación a proyectos de realidad aumentada e interfaces tangibles.

Comité Editorial de RIM

staff

RIM

Director
Prof. Carmelo Saitta

Secretario de redacción
Dr. Pablo Cetta

Comité editorial
Ing. Emiliano Causa
Dr. Pablo Cetta
Dr. Pablo Di Liscia
Mg. Raúl Lacabanne
Prof. Gumersindo Jerónimo de Jesús Serrano Gómez
Arq. Daniel Wolkowicz

Colaboran en este número
Carmelo Saitta
Pablo Cetta
Pablo Di Liscia
Emiliano Causa
Matías Romero Costas
Tarcisio Lucas Pirotta

Diseño
Arq. Daniel Wolkowicz

Corrección de textos
Rossana Cabrera

RIM es una publicación del Área
Transdepartamental de Artes Multimediales
del IUNA
Yatay 843, Ciudad Autónoma de Buenos Aires,
República Argentina

ISSN 1850-2954

Impreso en New Press Grupo Impresor S.A.
Paraguay 278, Avellaneda
Provincia de Buenos Aires
Marzo de 2011

índice

TARCISIO LUCAS PIROTTA

Es diseñador en Comunicación Visual y profesor en Producción Multimedial, diplomado en la Facultad de Bellas Artes, Universidad Nacional de La Plata.

Se desempeña como docente e investigador en la Facultad de Bellas Artes (UNLP) y en el Área Transdepartamental de Artes Multimediales (IUNA). Ha dictado cursos de posgrado y talleres relacionados con arte multimedia y nuevas tecnologías.

Como artista multimedia integra el Grupo Proyecto Biopus, desarrollando desde el año 2003 instalaciones interactivas que conjugan arte, ciencia y tecnología.

Técnicas de programación vinculadas a la realidad aumentada y a las interfaces tangibles

Tarcisio Lucas Pirotta

Palabras claves

Programación, realidad aumentada, interfaces tangibles, ArtoolKit, reactivision

1. Resumen

El presente trabajo tiene por objetivo analizar las principales herramientas de programación que permiten específicamente el reconocimiento de patrones para la creación de aplicaciones de realidad aumentada y la implementación de interfaces tangibles.

2. Generalidades

Para la elaboración de este artículo se ha tomado como objeto de análisis dos herramientas de programación, ArtoolKit y reacTiVision, vinculadas al desarrollo de aplicaciones de realidad aumentada y de interfaces tangibles respectivamente.

La primera consideración a tener en cuenta en la elección de estas herramientas y sus correspondientes técnicas de programación para el desarrollo de este escrito, es que han demostrado óptimos resultados en aplicaciones desarrolladas por integrantes de este proyecto de investigación, como por ejemplo la obra interactiva "*Mundo Circular*", y son hasta el momento unas de las mejores herramientas que se han implementado en diferentes proyectos a escala internacional. Por esto mismo es numerosa e interesante la cantidad de publicaciones y documentación que existe acerca de estas herramientas, lo que contribuye a realizar un correcto análisis y relevamiento de las técnicas de programación que emplean.

Otro aspecto importante es la especificidad de cada una de las herramientas, reactivision fue diseñada especialmente para el desarrollo de interfaces tangibles, más específicamente de mesas reactivas, mientras que Artoolkit permite desarrollar todo tipo de aplicaciones de realidad aumentada, excediendo aun aquellas donde se implementan interfaces tangibles.

Relacionado con esto último es interesante establecer brevemente cuál es la relación entre la realidad aumentada y las interfaces tangibles, y cuáles son las diferencias principales en las herramientas que permiten la creación de cada una de ellas.

Podemos considerar que el desarrollo de interfaces tangibles se encuentra siempre dentro del ámbito de las aplicaciones de realidad aumentada, son una variante que podríamos definir como un subconjunto del conjunto de realidad aumentada, lo cual nos permite deducir que todas las aplicaciones que involucran interfaces tangibles son consideradas realidad aumentada, aunque no sucede lo mismo de manera inversa, no todas las aplicaciones de realidad aumentada implican necesariamente la utilización de interfaces tangibles, sino que es un campo de acción mayor y que tiene mas posibilidades de alcance.

Por esto mismo si bien con la herramienta ARToolKit podemos desarrollar aplicaciones de realidad aumentada, lo que incluye el desarrollo de interfaces tangibles, con la herramienta reacTiVision solo podemos desarrollar interfaces tangibles.

La principal diferencia es el modo que cada una tiene para captar los patrones bitonales y detectar la posición del punto de vista del usuario, y de ahí las limitaciones y diferencias de cada una.

En las aplicaciones que emplean interfaces tangibles, como en el caso de las mesas reactivas, los patrones se encuentran siempre a la misma distancia de la cámara y dispuestas de manera perpendicular, por lo tanto, reacTiVision solo esta diseñada para captar patrones bitonales en 2D, o sea, patrones que solo se trasladan en relación con la cámara en los ejes X Y. Mientras que en las aplicaciones de realidad aumentada los patrones pueden estar dispuestos en cualquier lugar y posición, y pueden ser trasladados y rotados en cualquiera de sus ejes, por lo que las aplicaciones, como es el caso de ArtoolKit, deben ser capaces de captar los patrones con perspectivas en 3D.

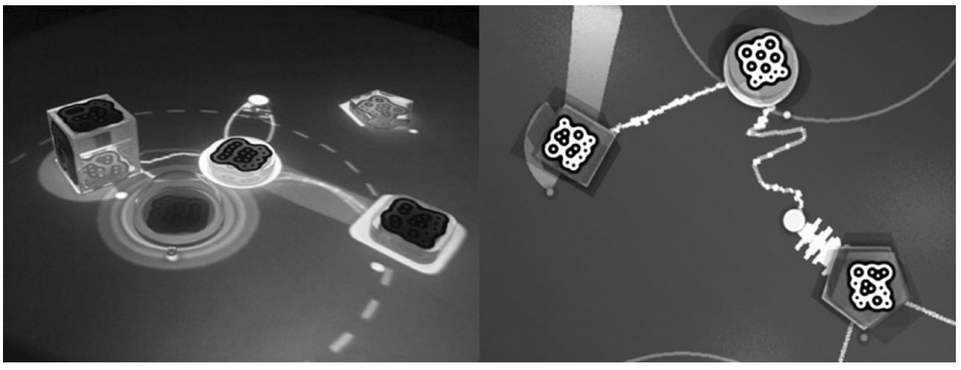

Figura 1: imágenes de la instalación reacTables desarrollada con reacTIVision (http://www.iua.upf.es/mtg/reactable)

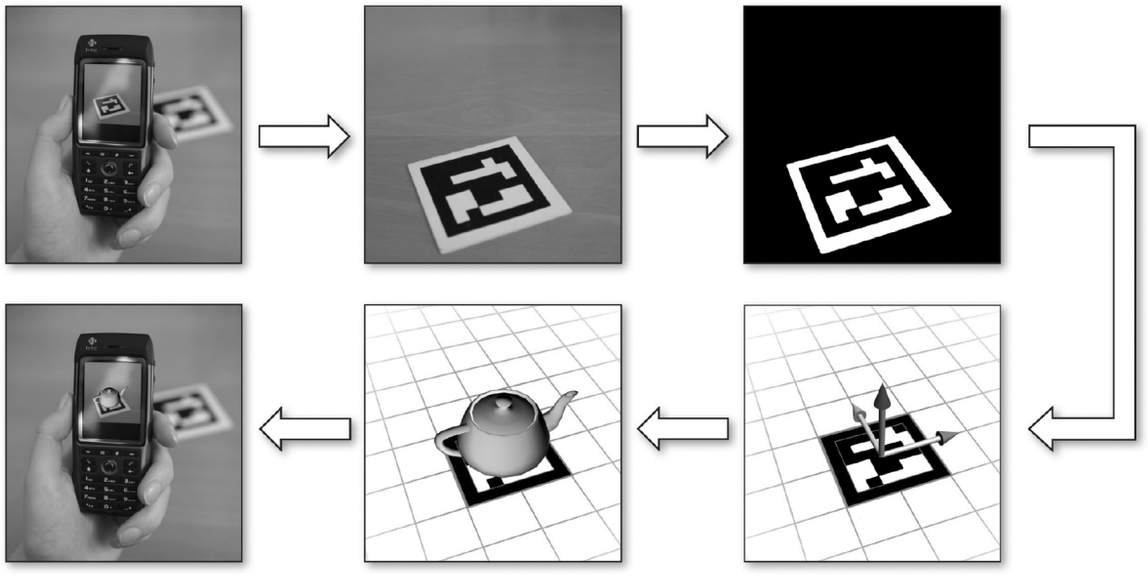

Figura 2: ARToolKit. imagen obtenida de http://www.icg.tu-raz.ac.at/Members/daniel/ARToolKitPlusMobilePoseTracking/download

3. ARToolKit

http://www.hitl.washington.edu/artoolkit/ ARToolKit es un conjunto de librerías para C/C++ para la construcción de aplicaciones de realidad aumentada que implican la superposición de imágenes virtuales con las del mundo real.

Como mencionamos anteriormente una de las dificultades en el desarrollo de este tipo de aplicaciones es el seguimiento del punto de vista del usuario. Para saber desde que perspectiva se construirá la imagen virtual, la aplicación necesita saber cómo el usuario está mirando en el mundo real.

ARToolKit utiliza algoritmos de visión artificial (o CV, Computer Vision) para resolver este problema, sus librerías de seguimiento de video, calculan en tiempo real la posición y orientación de la cámara en relación con las marcas ubicadas en diferentes objetos. Esto habilita el fácil desarrollo de un amplio rango de aplicaciones AR

3.1. Principales funciones

A continuación, analizaremos un programa básico implementado en ARToolkit que permite captar un patrón a través de una cámara y sobreimprimirle digitalmente un modelo 3D.

El esquema de funcionamiento lo vemos en el siguiente cuadro:

Etapa	Función
1. Inicio y config. de la aplicación	init
2. Captura de un cuadro de video	arVideoGetImage (llamada desde mainLoop)
3. Detección de Patrones	arDetectMarker (llamada desde mainLoop)
4. Cálculo de def. de cámara	arGetTransMat (llamada desde mainLoop)
5. Dibujo de los objetos virtuales	draw (llamada desde mainLoop)
6. Finaliza la captura de video	cleanup

Figura 3: esquema de funcionamiento de programa implementado en ArtoolKit

Las funciones más importantes son main, init, mainLoop, draw y cleanup, y serán explicadas en detalle a continuación.

3.1.1. Main

La función main es, como dice su nombre en inglés, la función principal del programa y la que llama y ejecuta a todas las demás funciones mencionadas en el párrafo anterior.

```
main(int argc, char *argv[])
{
init();
arVideoCapStart();
argMainLoop( NULL, keyEvent, mainLoop );
}
```

3.1.2. Init

La función init, declarada en la primera línea de la función main, contiene el código para comenzar la captura de video, leyendo los parámetros de la cámara, y definiendo la ventana de gráficos. Esto corresponde al paso 1 en el cuadro de la Figura 3. Luego, entramos en el estado de ejecución en tiempo real, con una llamada a la función arVideoCapStart, que es la que inicia específicamente la captura de video. A continuación, es llamada la funcion argMainLoop, la cual comienza el bucle con el programa principal, teniendo como argumentos a otra función llamada keyEvent, que detecta cualquier evento del teclado, y la función mainLoop que se asocia con el bucle principal de procesamiento de gráficos.

Como hemos mencionado, init es llamada desde la función main, y se utiliza para inicializar la captura de video y leer los parámetros iniciales de ArtoolKit.

En primer lugar, el dispositivo de video es abierto y se detecta el tamaño de la imagen, como vemos a continuación:

```
/* abre dispositivo de video */
if( arVideoOpen( vconf ) < 0 ) exit(0);

/* encuentra el tamaño de la imagen */
if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);
```

Luego es necesario inicializar los parámetros de ARToolKit. Los más importantes son:

- Los patrones a detectar que serán usados en la aplicación los objetos virtuales que corresponden a cada uno de esos patrones.

- Las características de la cámara utilizada.

Estos parámetros son leídos de archivos externos.

```
/* define parámetros iniciales */
   if( arParamLoad(cparaname, 1, &wparam) < 0 ) {
       printf("Error al detectar cámara !!\n");
       exit(0);
   }
```

Posteriormente se lee la definición del patrón en comparación con la información del patrón incluido en un archivo externo y se asocia el patrón con un identificador:

```
if( (patt_id=arLoadPatt(archivo)) < 0 ) {
    printf("error al leer patron !!\n");
    exit(0);
}
```

Por último, una ventana gráfica es abierta:

```
/* abre ventana gràfica */
argInit( &cparam, 1.0, 0, 0, 0, 0 );
```

3.1.3. MainLoop

La función mainLoop realiza la mayoría de las llamadas a funciones, abarcando desde el paso 2 al paso 5 del esquema del programa de la Figura 3.

Primero se captura un cuadro de video utilizando la función arVideoGetImage:

```
/* captura un cuadro de video */
if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
    arUtilSleep(2);
    return;
}
```

La imagen de video es luego visualizada en la pantalla con la función argDispImage, que tiene como argumentos la imagen capturada previamente con arVideoGetImage y la posición donde será mostrada:

```
argDispImage( dataPtr, 0,0 );
```

Luego la función arDetectMarker es utilizada para buscar dentro de la imagen cuadrados que puedan incluir los correspondientes patrones cargados anteriormente en la inicialización:

```
if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
    cleanup();
    exit(0);
});
```

El número de patrones encontrados es asignado a la variable marker_num, mientras que marker_info es un puntero a una lista de estructuras de patrones que

contienen la información de coordinación y reconocimiento, como así también los valores de id correspondientes a cada uno de los patrones.

En este punto del programa, la imagen ha sido mostrada y analizada, y a continuación todos los valores de los patrones detectados son comparados para asociar el patrón correcto y su correspondiente id:

```
/* chequea la visibilidad del patrón */
k = -1;

for( j = 0; j < marker_num; j++ ) {
    if( patt_id == marker_info[j].id ) {
        if( k == -1 ) k = j;
        else if( marker_info[k].cf < marker_
info[j].cf ) k = j;
    }
}

if( k == -1 ) {
    argSwapBuffers();
    return;
}
```

La deformación entre el patrón real impreso y la imagen que muestra la cámara puede ser detectada utilizando la función arGetTransMat:

```
/* determina la deformación entre el patrón real
y la imagen de la càmara */
arGetTransMat(&marker_info[k], patt_center,
patt_width, patt_trans);
```

La posición y orientación real de la cámara relativa al patron impreso es introducida en una matriz 3 x 4, patt_trans.

Finalmente, los objetos virtuales pueden ahora ser dibujados sobre la imagen del patrón empleando la función draw:

```
draw();
```

La función draw se divide en inicialización del render, configuración de la matriz y, por último, el render final del objeto.

Para inicializar el render es necesario definirlo en modo 3D y configurar unos mínimos estados de OpenGL:

```
argDrawMode3D();
argDraw3dCamera( 0, 0 );
glClearDepth( 1.0 );
glClear(GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
```

Luego de esto se requiere convertir los cálculos de deformación (matriz de 3 x 4) a un formato OpenGL a modo de arreglo de 16 valores, usando la función llamada argConvGlpara. Estos 16 valores corresponden a la posición y orientación real de la cámara, por lo cual al utilizarlos para definir los valores de la cámara virtual produce que cualquier objeto gráfico a dibujarse aparezca exactamente alineado con el patrón físico.

```
/* carga los datos de la matriz */
argConvGlpara(patt_trans, gl_para);
glMatrixMode(GL_MODELVIEW);
glLoadMatrixd( gl_para );
```

La posición de la cámara virtual es definida utilizando la función OpenGL glLoadMatrixd (gl_para). La última parte del código es la renderización del objeto 3D propiamente dicha. En el código a continuación se dibuja un cubo azul con una iluminación de color blanco:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_flash_shiny);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMatrixMode(GL_MODELVIEW);
glTranslatef( 0.0, 0.0, 25.0 );
glutSolidCube(50.0);
```

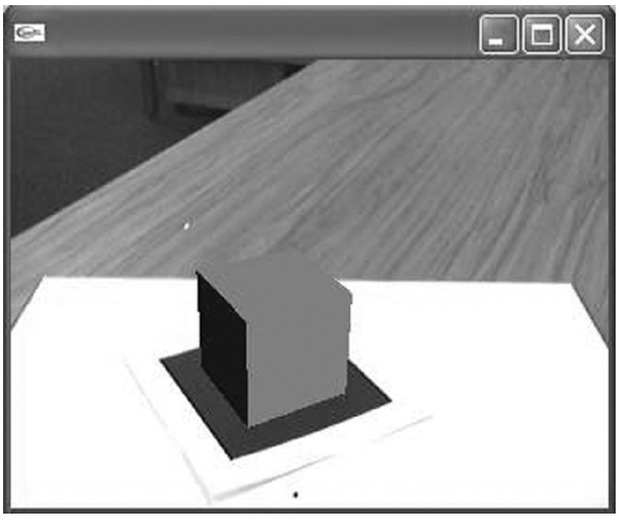

Figura 4: ejemplo de programa básico implementado en ArtoolKit
http://www.hitl.washington.edu/artoolkit/documentation/devstartup.htm

Estos pasos mencionados anteriormente (del 2 al 5 correspondientes al esquema de la Figura 3) se repiten indefinidamente mientras se ejecute la función mainLoop.

3.1.4. Cleanup

Al ser llamada la función cleanup, se detiene el procesamiento de video y se libera el disposivido de video, quedando disponible para otras aplicaciones:

```
arVideoCapStop();
arVideoClose();
argCleanup();
```

4. reacTIVision

http://reactivision.sourceforge.net/

Es un software de código abierto, un entorno para el reconocimiento rápido y efectivo de patrones en tiempo real.

Se diseñó principalmente como una herramienta para la creación de mesas tangibles a modo de interfaces. Basada en otras librerías, esta aplicación fue desarrollada por Martin Kaltenbrunner, en el Music Technology Group, en Barcelona, como parte del proyecto de reacTable, un instrumento musical que emplea las mesas tangibles anteriormente mencionadas.

Estas mesas tangibles están constituidas por una superficie translucida, sobre la cual se colocan diversos objetos que tienen la impronta de las diferentes marcas a reconocer (patrones).

Al ser ReacTiVision una aplicación ejecutable independiente, envía mensajes OSC (OpenSound Control) a través del puerto UDP, para lo cual se implementó el protocolo TUIO, especialmente diseñado para transmitir el estado de cada uno de los objetos. Es posible leer dichos parámetros desde otras plataformas utilizando una librería, como en el caso de TUIO client para la plataforma Processing

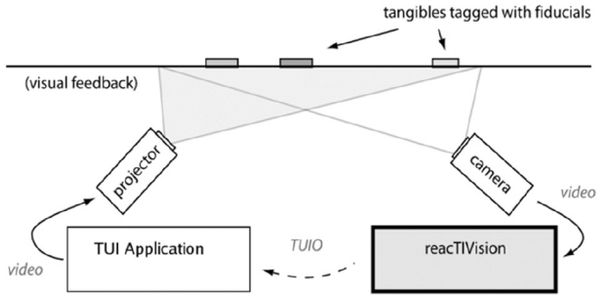

Figura 5: gráfico que describe el dispositivo empleado en reacTables (http://www.iua.upf.es/mtg/reactable)

4.1. Principales funciones

Tomando como referencia la librería anteriormente mencionada para Processing, a continuación mencionaremos los principales métodos y funciones necesarios para realizar una correcta lectura de los datos provenientes de reacTiVision y poder así construir una aplicación de interfaces tangibles.

En primer lugar es necesario crear una instancia cliente del tipo TuioProcessing client, de inmediato el cliente comienza a leer los mensajes provenientes de TUIO y genera eventos de nivel superior basados en los movimientos de los patrones o cursores.

```
TuioProcessing tuioClient = new TuioProcessing(this);
```

En este punto es importante destacar que reacTiVision no solo está diseñado para capturar patrones bitonales, sino que también puede detectar cursores, es decir, es capaz de captar las impresiones digitales a modo de puntos (dedos que se apoyan sobre la pantalla).

Cualquier aplicación que utilice reacTiVision para la captación de patrones o cursores necesitará implementar las siguientes funciones para ser capaz de recibir los eventos TUIO correctamente:

- addTuioObject (TuioObject tobj) se ejecuta cuando un nuevo patrón es detectado.
- removeTuioObject(TuioObject tobj) se ejecuta cuando un patrón ha sido eliminado.
- updateTuioObject(TuioObject tobj) se ejecuta para establecer el movimiento de un patrón, ya sea ubicación o rotación.
- addTuioCursor(TuioCursor tcur) se ejecuta cuando un nuevo cursor es detectado.
- removeTuioCursor(TuioCursor tcur) se ejecuta cuando un cursor ha sido eliminado.
- updateTuioCursor(TuioCursor tcur se ejecuta para establecer el movimiento de un cursor, ya sea ubicación o rotación.
- refresh(TuioTime bundleTime) actualiza continuamente la pantalla.

Cada TuioObject or TuioCursor está identificado con un único identificador, llamado SessionID, el cual se mantiene durante la existencia en pantalla del patrón.

A su vez, cada objeto de tipo patrón posee también un SymbolID que corresponde al número de patrón al cual está asociado (el diseño, la referencia gráfica impresa). En el caso de los objetos de tipo cursor, poseen un CursorID, el cual es siempre un número dentro del rango de todos los cursores detectados en ese momento.

Es posible obtener el valor de esos ID a través de los métodos getSessionID(), getSymbolID() o getCursorID().

Todas las referencias a los objetos son actualizadas automáticamente por el cliente TuioProcessing y siempre refieren a la misma instancia de los objetos correspondientes durante su tiempo de vida.

Por otro lado, los atributos de los objetos, tanto patrones como cursores, están encapsulados, y puede accederse a ellos utilizando métodos como los siguientes:

- getX() devuelve la posición en el eje x
- getY() devuelve la posición en el eje y
- getAngle() devuelve el ángulo de rotación

Existen otros métodos que también permiten obtener la información de los valores de velocidad o aceleración, como así también métodos adicionales muy convenientes para el cálculo de distancias o ángulos entre objetos.

El método getPath() devuelve un vector de TuioPoint que representa el trayecto de movimiento del objeto.

Finalmente, la librería contiene algunos métodos para la determinación de los estados de los patrones o cursores, ya sea de manera conjunta o individual.

- **getTuioObjects()** devuelve un vector con todos los patrones actualmente en pantalla.
- **getTuioCursors()**devuelve un vector con todos los cursores actualmente en pantalla.
- **getTuioObject(long s_id)** devuelve un patron específico o NULL según la presencia del mismo.
- **getTuioCursor(long s_id)** devuelve un cursor especifico o NULL según la presencia del mismo.

5. Conclusión

Luego del recorrido realizado por las técnicas de programación descriptas, es posible concluir en primer lugar que el avance tecnológico de los últimos años ha favorecido el desarrollo de estas herramientas y contribuido a una redefinición de las interfaces, como así también de la simplificación en la implementación de estas herramientas que permiten el desarrollo de aplicaciones de realidad aumentada que hace algunos años hubieran parecido imposible de poner en práctica.

La utilización de estas herramientas en el proceso de creación artística y educativo es un aporte positivo, ya que pone a disposición de artistas y educadores un recurso más de expresión y comunicación, donde la tecnología contribuye a generar sentido y, en cierta medida, forma parte del discurso.

Referencias bibliográficas

Causa, Emiliano. *Desarrollo de una Aplicación con Interfaces Tangibles* (2008), <http://www.biopus.com.ar>.

<http://www.arduino.cc>.

<http://artoolkit.sourceforge.net/apidoc/files.html>.

<http://www.hitl.washington.edu/artoolkit/documentation>.
<http://mtg.upf.es/reactable/>.

<http://www.panda3d.org/apiref.php?page=ARToolKit>.

<http://www.processing.org>.

<http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php>.

<http://www.tuio.org/?processing>.
.